Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 327 (2016) 125–146

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Efficient Image Stitching through Mobile Offloading

Qiushi Wang [1] Fabian Reimeier[2] Katinka Wolter[3]

Institute for Computer Science
FU Berlin
Berlin, Germany

Abstract

Image stitching is the task of combining images with overlapping parts to one big image. It needs a sequence of complex computation steps, especially the execution on a mobile device can take long and consume a lot of energy. Mobile offloading may alleviate those problems as it aims at improving performance and saving energy when executing complex applications on mobile devices. In this paper we investigate to which extent mobile offloading may improve the performance and energy efficiency of image stitching on mobile devices. We demonstrate our approach by stitching two or four images, but the process can be easily extended to an arbitrary number of images.
We study three methods to offload parts of the computation to a resourceful server and evaluate them using several metrics. For the first offloading strategy all contributing images are sent, processed and the combined image is returned. For the second strategy images are offloaded, but not all stitching steps are executed on the remote server, and a smaller XML file is returned to the mobile client. The XML file contains a homography information which is needed by the mobile device to perform the last stitching step, the combination of the images. For the third strategy the images are transformed into grey scale before being transmitted to the server and an XML file is returned. The considered metrics are the execution time, the size of data to be transmitted and the memory usage. We find that the first strategy achieves the lowest total execution time but it requires more data to be transmitted than both the other strategies.

Keywords: Image stitching, Mobile offloading, Program Partitioning

1 Introduction

Taking photographs has become one of the most widely used applications on mobile devices. Cameras in mobile phones have become very good over the past years. Images are used to preserve memorable moments, but also to take scans of documents. For panoramic pictures as well as for images of larger documents one image alone may sometimes not be enough to represent the full picture. Then several images of parts of the scene are taken. To restore the full picture those images must be

[1] Email: qiushi.wang@fu-berlin.de
[2] Email: f.reimeier@fu-berlin.de
[3] Email: katinka.wolter@fu-berlin.de

http://dx.doi.org/10.1016/j.entcs.2016.09.027

combined by stitching them in the overlapping sections [16]. While this stitching of several images may be desirable also for environmental documentation purposes, the computations needed to stitch several images are rather extensive [13].

Even though the user of the mobile device produces the individual pictures and requests to obtain the combined image, the stitching itself need not be performed by the mobile device. Since the image stitching algorithms include heavy computation this work would ideally be performed by a powerful server, not by a mobile device. The execution of complex tasks on powerful remote devices can save time and extend the battery life of the mobile device, but the required data for the computation has to be transmitted to the server which after completion returns the results back to the mobile device.

Mobile application offloading is a technique that transfers heavy computation to a server to reduce the power consumption of mobile devices. In some cases completion of the task is faster when migrating the computation to a server. Different offloading strategies have been explored in [15,26,27]. Some requirements must be fulfilled for mobile application offloading to be beneficial. The transmitted data must not be too large and the network connection has to be sufficiently large and stable. The objective of this paper is to explore the potential benefits of mobile application offloading for image stitching.

We have implemented a local version of the image stitching algorithm as a mobile application which can use three different strategies to offload either the full computation or parts of it. Image stitching consists of five different steps, some of which need more processing than others. We used the execution time as the metric to measure the complexity of the five steps. The measurement results show that the computation in the first and last step are much heavier than for the other steps. As the image files are required for each step, once the computation is offloaded, it should stay on the server side as long as possible. Accordingly, offloading strategy 1 transfers the whole stitching process to the server while strategy 2 and 3 compute the last step locally in the mobile device. In order to reduce the size of the images to be transmitted, strategy 3 compresses the images into grayscale before sending them to the server. The performance of the three strategies is compared using experiments and we find strategy 1 to perform best according to our metric.

The remainder of this paper is organized as follows, in the next section we briefly introduce the method of image stitching and recapitulate the background of mobile offloading. In Section 3 the steps of stitching two images together are introduced, which can be easily extended to stitching more than two images. Then, in Section 4, three options of offloading different parts of the image stitching procedure are proposed. In Section 5 the experiment configuration and the metrics to evaluate the different methods are shown. In Section 6, the experiments with the different offloading methods are evaluated, the results are compared and discussed. The last section concludes this work.

2 Background and Related Work

2.1 Image Stitching

Image stitching is an old problem in computer vision that has been solved in different ways in history and is often described in the literature [6,24,13,21,16]. The process of combining different images to one is used, for example to combine images taken by satellites for navigation, to generate panoramic views from impressive landscapes and huge objects [24], the stitching of microscopical scenes [13] or for image stabilization [24]. Today, taking pictures has become one of the most widely used applications for mobile phones, hence stitching those images has also gained relevance.

There are two main categories of methods for automatic image stitching, direct and feature-based methods. Direct methods are using all image data and minimize the pixel-to-pixel dissimilarities. Feature-based methods extract features from the images and try to match these, which enables the automatic detection of the correlation of images with overlapping parts [6,24].

There exist several applications for mobile devices which are able to do image stitching, like *AutoStitch* [1] for iOS or *DMD Panorama* [2] for Android, but none of them uses mobile offloading for increased performance.

To create a prototype of an image stitching application that uses mobile offloading a very basic, simple and not scale invariant feature-based approach is used, which basically can be divided into five steps, feature-point detection [11,17,23], feature-point description [17], feature matching [17], homography estimation [4,12,21] and warping and projection. Those steps are briefly described in Section 3. For all these steps intensive research has been done and there are several ways of doing them [6,13,24]. Herbon et al. [13] have also described methods to reduce the complexity of the image stitching procedure for the execution on mobile devices by adding preconditions. The implementation of the used algorithms in this paper is not optimized for mobile devices, does not include image post-processing, like blending or feathering to hide the transitions from image to image [16], and does not deal with ghosting effects.

2.2 Mobile Offloading

For the sake of making applications on mobile devices fast and energy-efficient, the concept of mobile application offloading has been around for more than a decade. Several methods (CloneCloud, MAUI, Cloudlets, among others) have been proposed to support the seamless use of augmented computation to a mobile device. Most of the known methods are architectures that aim to provide a comfortable development environment for the programmers, who want to implement their mobile applications with offloading.

Research in offloading methods can be divided into three main directions [10]: client-server communication, virtual machine migration and mobile agents. Among the three categories, the virtual machine migration is the most popular one and it

represents the future development trend. The great advantage of a virtual machine is the high consistency of the two version applications on the mobile side and on the server side. Since nearly no modification is required to move the application from the mobile device to the server, the burden on the programmer is reduced. Although the client-server communication supports a more robust link between both sides, the application has to be pre-installed. The main drawback of mobile agents is the expensive cost of agent management and application synchronization. For understanding the detailed properties of each category, we make a deep exploration into the specific structures. As the virtual machine has attracted most attention, we first list four sample structures belonging to this category below:

CloneCloud boosts unmodified mobile applications by offloading the right portion of their execution onto device clones operating in a computational cloud [7]. It proposes a very interesting idea: to continuously have a synchronized copy of the mobile device contents in the cloud in order to offload operations like file searches or virus scans. MAUI achieves the benefits of maximizing the potential for energy saving through fine-grained code offload, while minimizing the changes required to applications [9]. It applies the managed code environments to dynamically partition programs according to profiling and serialization of an application. VM-Based Cloudlets exploit virtual machine (VM) technology to rapidly instantiate customized service software on a nearby cloudlet and use that service over a wireless LAN [22]. Due to the physical proximity between cloudlets and mobile clients, the one-hop network latency brings the advantage of transient connections at the cost of a large-scale implementation with high price. Mobicloud treats mobile devices as service nodes and organizes them in an ad-hoc network. Each node is mirrored in the cloud and used as a virtual component to provide computation service [14]. In addition to providing traditional computation services, Mobicloud enhances communication by addressing trust management, secure routing, and risk management issues in the network.

These platforms can be considered as the prototypes of a general and standard mobile application offloading structure. We believe that mobile application offloading systems are still in an early stage of development. With the invention of more intelligent mobile devices, mobile offloading structures will also be further developed. There is no doubt, more novel structures will also be proposed. In fact, if some uniform standards can be formulated to define either the offloading work flow or the basic elements of the system structure, the development of mobile offloading will be facilitated.

3 Stitching Procedure

The task of image stitching is to paste two or more given images together such that points which are shown on both images are on top of each other [21]. This problem has been solved in different ways [6,24]. In this case, a feature-based approach is used, because this approach is more robust according to scene movement and it enables the automatic detection of the correlation of images with overlapping parts

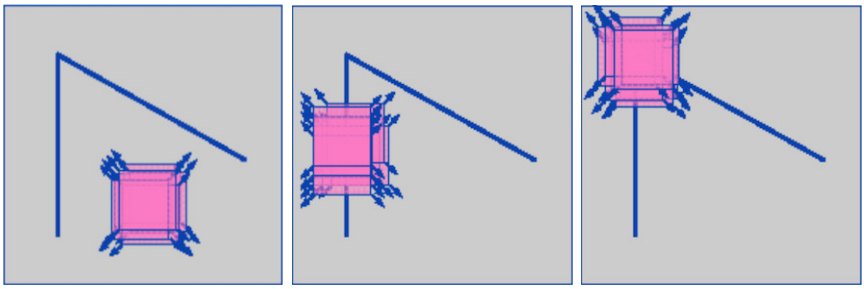

Fig. 1. The idea of Moravec's corner detection [8].

[24].

We assume that images which should be stitched together are taken using the same camera, with identical zoom settings, and that the camera was only rotated around the center of its lense while taking the pictures. Images taken by a camera which only rotates around the center of its lense are related to each other by non-singular 3×3 matrices called homographies [12]. If there are at least four corresponding points in two given images it is possible to compute a homography which maps one image onto the other. Warping one of the image according to this homography and pasting it onto the other will create a panoramic image and will therefore solve the image stitching problem [21].

3.1 Feature-point Detection

The first step in image stitching is to decide which type of points will be used to find correspondence between two images. One possible option is to use corners as feature-points of the images [11]. To detect corners in an image the *Harris Corner Detection algorithm* [11] can be used, which is rotation invariant and robust to illumination changes but not scale invariant [11,17,23].

The *Harris Corner Detection* is based on *Moracec's Corner Detection algorithm* [19] and, like its predecessor, uses small windows around each pixel. These windows are slightly shifted in every direction. The change of average image intensity resulting from those shifts is used to make a statement about the existence of a corner in the area of the pixel that is located in the center of the considered window. The image intensity of a pixel can be defined as the greylevel of the pixel in the 8-bit grayscale version of the image.

If the change of intensity is very small, the pixel is located in a flat region. If the change of intensity is small along one direction and large along the others, the pixel is part of an edge. If the change of intensity is large in all directions, the pixel is part of an isolated point or a corner [11]. These three possible cases are shown in Figure 1.

Chris Harris and Mike Stephens [11] proposed the following corner response function, which can be used to decide if a pixel is in the area of a corner or not:

$$R(x,y) = Det(M(x,y)) - k \cdot Trace(M(x,y))^2$$

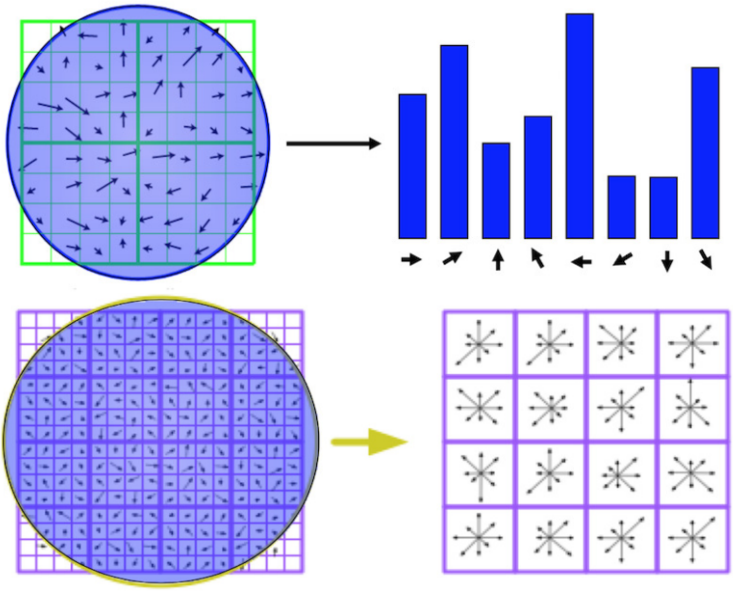

Fig. 2. Creation of orientation histograms and descriptor vectors [20].

where

$$M(x,y) = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}.$$

The matrix M is built out of the weighted sum of image gradients in x direction, I_x, and y direction, I_y, of all pixels in a window around the pixel at position (x, y). These derivatives can be obtained by applying an edge detector like the *Sobel Operator* [5] on the images.

If the corner response is negative, the pixel is part of an edge. If the response is small but positive, the pixel is part of a flat region and if it is positive and large, the pixel is part of an isolated point or a corner [11].

To find corners, R has to be evaluated for each pixel of all images. Then the local maxima of R in each image have to be found, which are then considered the corners and therefore the feature-points of the image.

3.2 Feature Description

The feature-points described in Section 3.1 are single pixels. The only information we can get when considering only one single pixel is its (x, y) position and its color. Because the images to stitch images together can differ, the position of corresponding points do not have to be identical on both images and the illumination might have changed, so that the color is also not the same [23]. Therefore, it is not sufficient to consider only the found feature-points to compare and find corresponding ones. In order to compare them, they have to be described uniformly with the help of more information about the area around them [17].

The SIFT descriptor is part of the *Scale Invariant Feature Transform* approach

by David G. Lowe [17] and uses histograms of gradient directions and magnitudes around SIFT keypoints to describe them and generate SIFT features. To provide the scale invariance, SIFT keypoints provide information about the scale of the image at which the keypoint was detected [17]. The feature-points described in Section 3.1 do not provide this information, but because of the assumption that all images are of the same scale, the scale invariance is not required. The descriptions of the feature-points created by the SIFT descriptor will therefore not be scale invariant, but the rotation invariance and partial illumination invariance will persist, because the scale information of a SIFT keypoint is only used once in the description process to set a scaled version of the original image as basis for the description [17].

In order to provide rotation invariance of the description, a main orientation has to be assigned to each feature-point [17]. This is done by using a window around each feature-point and the generation of an orientation histogram of gradient directions of the pixels in the area of the window, as can be seen in Figure 2.

For each pixel the magnitude of its image gradient is added to the corresponding bucket of the orientation histogram. The orientation that belongs to the highest peak in the histogram will be assigned as main orientation of the feature-point [17]. To generate the description of a feature-point, a bigger window around the feature-point is taken, which is rotated by the main orientation and divided into smaller windows. Then for each of the small windows an orientation histogram is created just as described before. These histograms are collected in a vector, which is now the description of the feature-point, a feature [17]. In Figure 2 the process of creating the description vector can be seen, the circles represent a gaussian weight function, which can be used to improve the results [17].

3.3 Feature Matching

The next step is to match the features of the different images and therefore to find corresponding points. Because the considered features are vectors as described in Section 3.2, a method to measure the similarity of vectors is needed.

One method to measure the similarity of two vectors is to compute their Euclidean distance [17]. If the distance is small, the vectors are very similar. If the distance is large, they differ much. One very intuitive and simple way to find the best match for a feature f_A of image A in the set of features of image B is to find the feature f_B in image B which has the smallest Euclidean distance to f_A [17]. This approach finds a best match for every feature f_A of image A, if the set of features of image B is not empty. As setting a global threshold for the distance of features to filter false matches does not perform well [17] a threshold for the ratio of the distance of the feature f_A to the nearest feature $f_{B,best}$ and the second nearest feature $f_{B,2ndbest}$ found is used. If the distance to the best matching feature is much smaller than to the second best match, the best match is found with high certainty [17].

3.4 Homography Estimation

With the corresponding features found in Section 3.3 it is now possible to compute a homography H through the *Direct Linear Transformation algorithm (DLT)* described in [12,4], such that for all corresponding feature-points in homogeneous coordinates p and p'

$$H \cdot p = p'.$$

Because four correspondences are sufficient for computing such a homography, the mapping may not be distinct. To find a very good fitting homography to map one image onto the other, an adapted *Random Sample Consensus algorithm (RanSaC)* can be used [18].

3.5 Warping and Projection

With the homography H found as described in Section 3.4 it is possible to paste the two images onto each other. To do so, a new empty surface is created and one image is put in the center of it. Then the positions of the pixel of the other image on the new surface can be computed by multiplying the coordinates of the pixel in the original image with H.

4 Implementation of Offloading for Image Stitching

Mobile offloading addresses the problem of providing performance and saving energy when executing complex applications on mobile devices with limited hardware resources and available energy by sending heavy computation to resourceful servers with no issues according to battery lifetime and afterwards receiving the results from them [26,27]. This simple basic idea is illustrated in Figure 3.

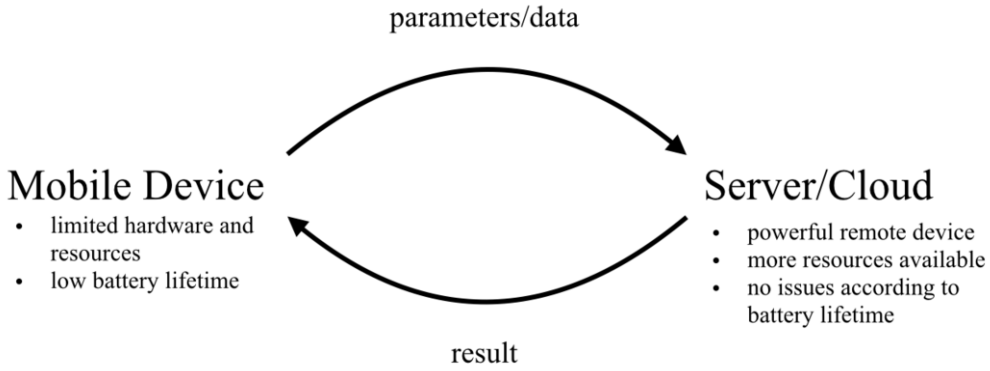

Fig. 3. Basic idea of Mobile Offloading.

To develop different offloading methods, the first step is to determine where the most expensive computation in the stitching process is located. To that end we have taken measurements of the local processing time split up by phase of the image stitching process. Figure 4 shows the average execution time of the different steps of the image stitching process on a *LG Nexus 5* smartphone.

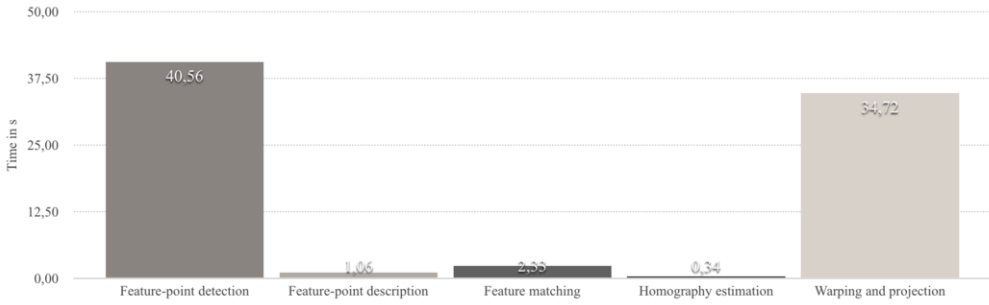

Fig. 4. Execution time of the image stitching steps for two images.

It can be clearly seen, that steps one and five, feature-point detection as well as warping and projection are the heaviest tasks by far for the mobile device. So these tasks should be offloaded to the powerful remote server. The feature-point description, feature matching and homography estimation take less time, because these steps are only applied on the detected feature-points and their close surrounding. The amount of feature-points is obviously much smaller than the total amount of pixels of all images. On the other hand for finding the corners or the position of a pixel in another picture all pixels have to be considered. Assuming that processing on the server is faster than on the mobile device, once the algorithm is running on the server, as many processing steps as possible should be performed there in order to minimise the local execution period. For image stitching it makes no sense to return the execution back to the mobile device and send it to the server again. Because frequent data transmission between the mobile client and the server consumes much time and energy if should be avoided as far as possible. The basic structure the offloading methods should have is shown in Figure 5.

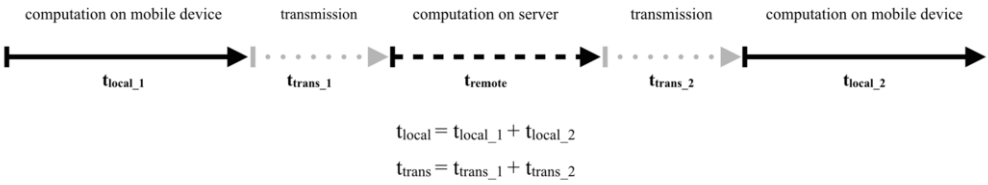

Fig. 5. Basic structure of an offloading method for image stitching.

The question now is when to start offloading and when to stop and return processing to the mobile device. We study three methods to find the optimal solution.

4.1 Method 1 - Send images, Receive image

The first method is to offload the complete stitching procedure hoping to minimize the total execution time, the local execution period and the memory usage. The images will be sent to the server directly and the resulting image will be returned by the server. The mobile device only has to send the images, receive the result and present it on the screen as can be seen in Figure 6, method 1. The local execution

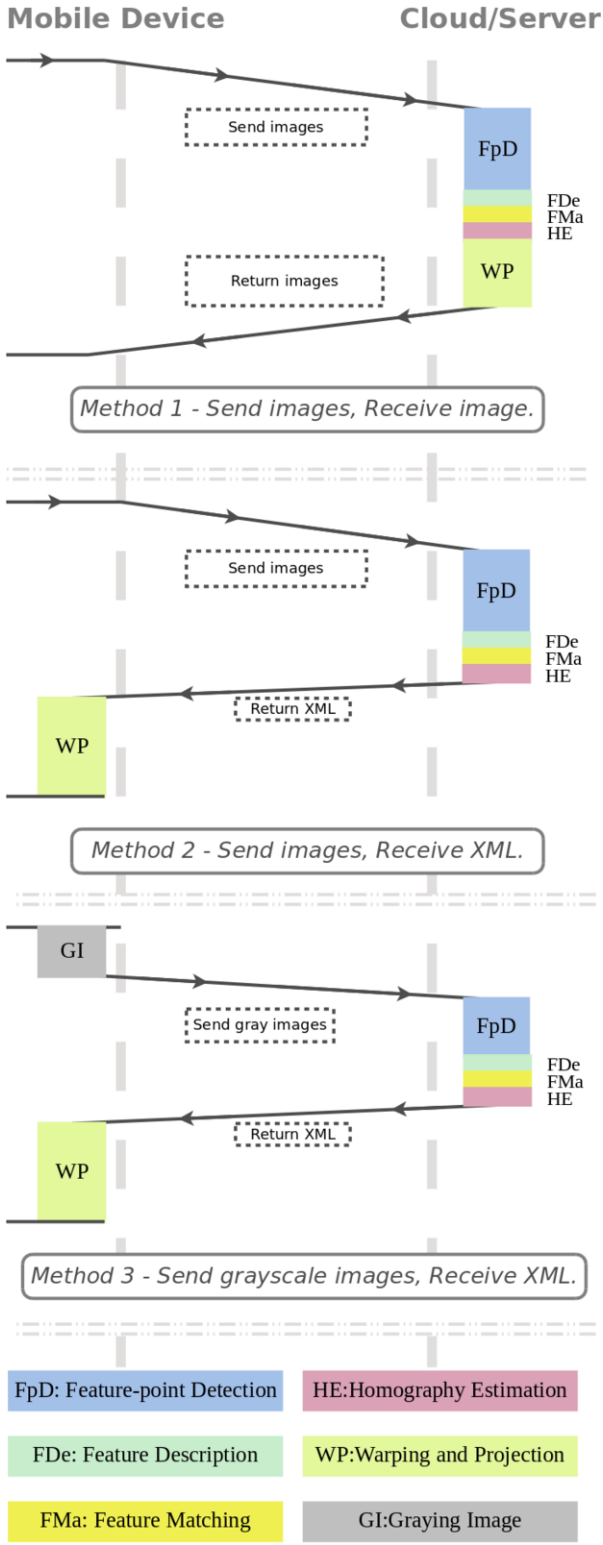

Fig. 6. Three offloading methods

time will be close to zero, because no part of the image stitching process is done on the mobile device.

The time to send all the images to the server and the resulting large image back to the client is a main part of the costs of the method in terms of time, so the transmission times to send and receive are expected to be large. This method is the only solution to offload the step of warping and projection to the server. To warp and project the images, the complete image data is required on the server. If other parts of the stitching process would be computed on the mobile device before offloading, this information and the images have to be transmitted to the server. So starting the offloading later and stopping at the end of the projection would lead to an increased transmission time and local execution period and no obvious advantages would be expected.

4.2 Method 2 - Send images, Receive XML

The second method, as shown in Figure 6, is an approach to minimize the total execution time and the data return time, respectively, as the amount of data to send from the server to the client is reduced. As the image data is originally located on the mobile device, it does not have to be sent back from the server. To stitch the images together only the order of the images and the homographies, which map the images onto each other, are needed.

This information can be sent as XML. The XML data will be much smaller than the large panoramic image and therefore the data transmission time will be reduced. After parsing the XML data, the mobile device can stitch the images together locally, according to the received order and the homographies. Since the last step of the image stitching process will be performed on the mobile device, the local execution period will be slightly larger, but the time to send the result back to the server is expected to be much smaller than in the first method.

4.3 Method 3 - Send grayscale images, Receive XML

The third method aims at minimizing the total execution time and the transmission time by minimizing both file sizes of the data to send and return. If the images are converted into 8-bit grayscale images, the time needed to compute the image intensities in the process of image stitching can be reduced. Because the color information is not needed, these images need less memory than color images. For that reason, the time needed to send them to the server will also be less than sending the coloured ones.

As the server does not have the color information, it is not possible to send the final stitched image back. For that reason, again only the image order and the homographies are sent back as XML data. Since the last step and part of the first step of the image stitching process are executed on the mobile device, as shown in Figure 6 method 3, the local execution period can be expected to be the largest among the three methods. The transmission times will probably be the smallest.

4.4 Comparison of the three offloading policies

According to the described methods the most relevant aspect is to find a good trade-off between local execution time and data transmission times. Roughly speaking, by decreasing the transmission times, the local execution time is increased and by increasing the transmission times, the local execution time can be decreased.

Across different network speeds the changes in the local execution time will be approximately constant for each method when stitching the same images, but the time to transmit the data hugely depends on the network speed. If the network speed is low, the reduction of the data size to send in method two and three will be leading to a large decrease of transmission time, whereas in the case of a fast network, the decrease is expected to be less relevant. So if the network connection of the mobile device is fast and stable, the first method is expected to be the best one according to the total execution time because the resource-constrained mobile device is little utilised.

If the network connection is slow and not very stable the third policy is expected to be best, because it benefits much from the reduced amount of data to transmit. The second method might also be better than the first one in this case. If the data to transmit should be minimized, the third method is expeted to perform best in all cases.

The memory usage on the mobile device should be minimized when using the first method and the second and third policy will also use less memory than the completely local execution of the algorithm.

In general, we expect that all the described methods should be better than completely local execution according to their total execution time, local execution period and memory usage.

5 Experiment Configuration

5.1 Platform Structure

We have implemented a standard local Java application for image stitching. The only library used besides the standard ones was the *Jama*-library [3] for matrix operations like matrix multiplication, inversion and computation of the singular value decomposition. The Java application was adapted to run as an Android application on a mobile device. The offloading methods introduced in [26] were added to the Android application later. Then the image stitching application was migrated to the remote Apache Tomcat server. A new webpage was added to the website [4] to integrate the new application into our existing web application [25]. A Java servlet was developed to cover all the different requests of the Android client according to the offloading methods.

The Android application was tested on a *LG Nexus 5* smartphone with Android 5.1.1 Lollipop (API 22) as operating system. The smartphone was connected to a

[4] https://www.mi.fu-berlin.de/offload/

computer running Android Studio 1.2.2 to measure the memory usage. The internet connection was provided by a hotspot to a MacBook Pro, which was connected by LAN to the internet. By doing so it was possible to emulate different network conditions by using the program *Network Link Conditioner* available through XCode. In the experiment, we emulated two kinds of network conditions, DSL (2MB/s download, 256KB/s upload, 5ms delay) and Edge (240KB/s download, 200KB/s upload, 400ms delay). The basic setup of the experiment can be seen in Figure 7.

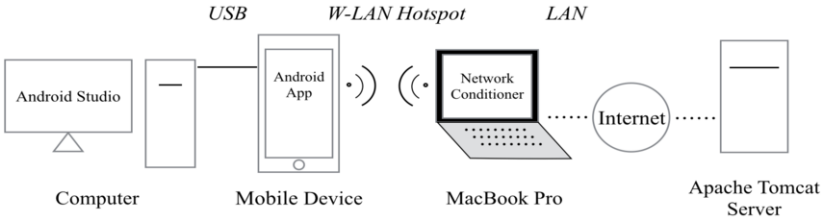

Fig. 7. The Experiment Configuration.

The application was tested with two and four images. The execution was repeated 20 times for each combination. The time stamps for measuring were automatically written to a *.csv* file stored in the smartphone at the end of the computation.

Some problems occurred when developing the Android application. The images taken by the camera of the *LG Nexus 5* have a high resolution of 3200×2368 pixels and required around 30 MB of memory each when loaded into the application. Although the standard local Java application running on a laptop was capable of stitching the high-resolution images, it was not possible to stitch them together on the smartphone. Because the available heap size for the Android application was too small and out of memory exceptions were thrown.

Improving the memory efficiency of the Android application finally avoided out of memory exceptions when stitching high resolution images, but the computation took extremely long compared with the standard application on a more powerful desktop PC or laptop. Before wider deployment of our methods further improvements of memory management are needed.

In this work we solved the memory problems using a simple solution. Before running the application, the images were scaled down to a lower resolution automatically. This reduced the memory requirements of the whole process and it is used for all results presented in Section 6.

5.2 Considered Metrics

As the different offloading methods for image stitching should be compared, metrics are needed to evaluate them.

Total execution time: The first metric is the total execution time $T_{offload}$ when using an offloading method. According to the performance metric, a program should be offloaded if the total execution time needed to compute the whole

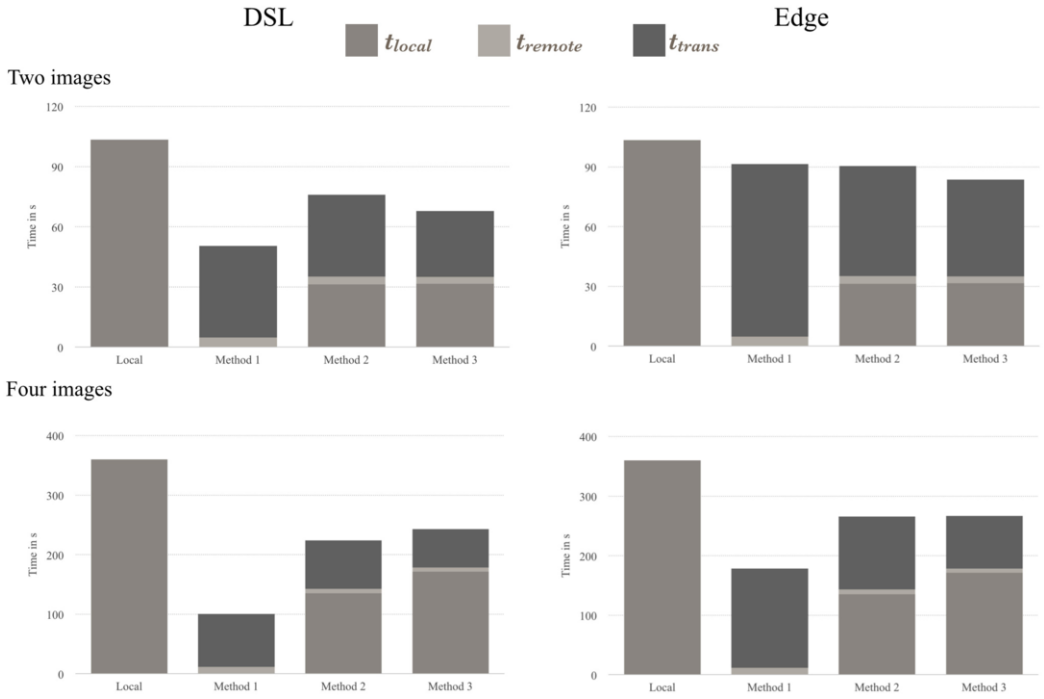

Fig. 8. Total Execution Times.

program on the mobile device is greater than the total execution time needed when using mobile offloading, i.e. $T_{local} > T_{offload}$.

Local execution time: The second metric is the local execution time t_{local}. This is the time needed to compute parts of the image stitching process on the mobile device. As it is assumed that the computation on the server is more efficient and saves the battery life of the mobile device, the maximum possible amount of computation should be performed on the server. Therefore the local execution time should be as small as possible.

Remote execution time The remote execution time t_{remote} will also be measured. It is the time needed to compute parts of the image stitching process on the server.

Transmission time: The transmission time t_{trans} consists of the time to send the required data to the server $t_{toServer}$ and the time to return back the results to the client $t_{toClient}$. The transmission time should be minimized, because sending data can be very expensive for mobile device users when they are in roaming state. In addition, under poor network condition the transmission is often slow and unstable [26].

Amount of data to send: By looking at monthly mobile phone plans, internet flat rates for mobile devices often have monthly data volume limits. So the amount of data to send to the server $d_{toServer}$ and return to the client $d_{toClient}$ should also

be measured and minimised.

Memory usage: The last metric used in this paper is the memory usage m of the mobile device. The memory of mobile devices is very limited and therefore it usage should be minimised.

As introduced in Section 4.4, to measure the benefit of trading transmission time against local execution time according to the total execution time $T_{offload}$, the ratio of their change can be used.

$$b = \frac{\Delta t_{local}}{\Delta t_{trans}}$$

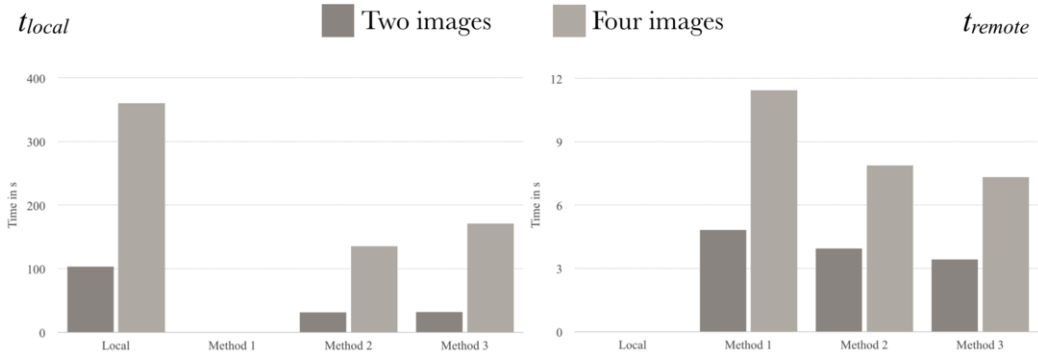

Fig. 9. Local and Remote Execution Period t_{local} and t_{remote}.

If $b < 1$, the tradeoff leads to a decrease of $T_{offload}$. If $b = 1$ the tradeoff will not influence the total execution time but can be desirable, for example if there are limits to the monthly data volume and the reduce of transmission time is therefore better than less local execution time. If $b > 1$, the tradeoff will increase the total execution time.

6 Measurement

In the following, the three proposed methods will be measured according to the metrics from Section 5.2.

6.1 Total Execution Time

The total execution time $T_{offload}$ of the proposed methods equals the sum of the local execution period t_{local}, the transmission time t_{trans} and the remote execution period t_{remote}. At first, the mobile device was connected via WiFi to the Internet. The accumulated resulting total execution times are shown in Figure 8.

It can be seen that all offloading methods have shorter execution times than the completely local execution. Offloading method one described in Section 4.1 leads to the shortest execution time. By using this method, the time needed to stitch two images is decreased on average by 50%. For stitching four images, around 73%

less time is needed. The advantage of method one over the other methods increases with the increasing amount of images to stitch.

The biggest parts of the execution with method two and three are the local execution time t_{local} and the transmission time t_{trans}. The remote execution period is very small, because the remote server is very powerful. It can also be seen, that trading the transmission time t_{trans} against local execution time t_{local} is not beneficial, if a good Internet connection exists. The local execution time needed to warp and project the images is larger than the saved transmission time and therefore $\frac{\Delta t_{local}}{\Delta t_{trans}} > 1$.

Method three gives a better result than method two when stitching two images whereas the stitching of four images does not benefit from the precomputing of the images on the mobile device. The amount of local execution time t_{local} needed to convert the images to 8-bit grayscale images grows faster than the decrease of the transmission time t_{trans} through pre-computation.

The total execution time of all offloading methods was increased by switching to the slower data network Edge, as can be seen in Figure 8. Under poorer network conditions the methods two and three are slightly faster than method one when stitching two images, but method one remains faster when stitching four images. The reason is that the local execution time reacts more to the addition of images than method two and three can compensate by reducing the transmission time t_{trans}. Nevertheless the difference of the total execution times of method one and method two and three to stitch four images is decreased in the case of lower network speed.

6.2 Local Execution Time

In Figure 9 the local execution time of the methods are shown. Using method one sets the local execution time to zero, because no local computation takes place. But when using method two, as the warping and projection have to be done on the mobile device, local execution time takes longer. The third method results in the highest value of t_{local} in comparison to the other offloading methods because the compression of the images has to be performed on the mobile device as well. Nevertheless the local execution time of method three is lower than the local execution time of computing the whole stitching process locally.

The local execution time is obviously not affected by network conditions, but t_{local} of method two and three increases when more images are stitched. The local execution time of method one is neither affected by network conditions nor by the number of images to stitch.

6.3 Remote Execution Time

In Figure 9 the remote execution time of the different offloading methods are shown. As well as the local execution times, they are independent of network conditions, but are affected by the number of images to stitch. Method one leads to the highest remote execution time, because all the computation is offloaded to the server. We

assume a faster server than in the mobile device and therefore the pure image stitching process takes on average 96% less time.

Method two keeps the step of warping and projection on the mobile device, therefore the remote execution time is shorter. Method three also keeps the conversion of the images to 8-bit grayscale images on the mobile device, which reduces the remote execution time a bit more. Since the part of warping and projection is more computationally intensive than the conversion, the difference of the remote execution time of method one and two is greater than the difference between the ones of method two and three.

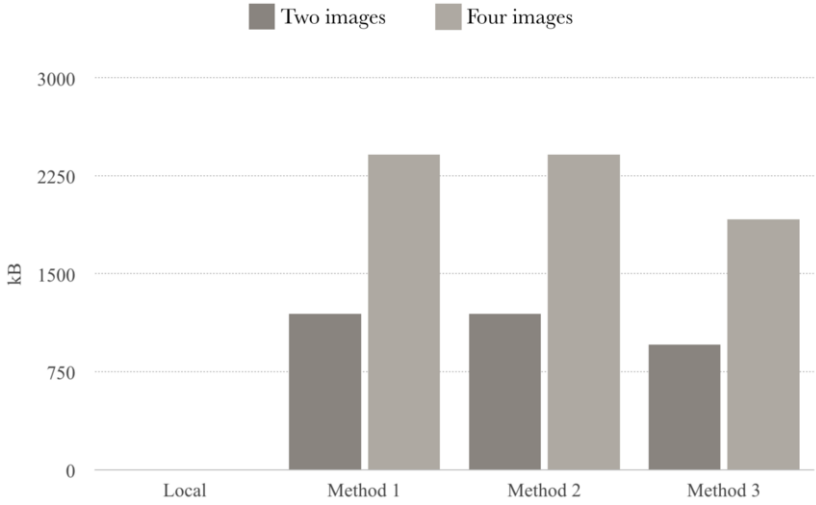

Fig. 10. Size of Data to send to the Server $d_{toServer}$.

	$d_{toClient}$ (Bytes) (2)	$d_{toClient}$ (Bytes) (4)
Local	0	0
Method 1	810595	1333806
Method 2	1250	2117
Method 3	1250	2116

Fig. 11. Size of Data to send to the Client $d_{toClient}$.

6.4 Transmission Time

The transmission time t_{trans} depends on the amount of data to send to the server and to return to the client as well as on the network conditions. With increasing amount of data to send, data transmission takes longer and the same amount of data is obviously transmitted slower with decreasing network speed.

The methods one and two send the whole image data, compressed to JPEG format, to the server. Method three compresses the image data by omitting the color information. Therefore only one byte per pixel has to be transmitted. However, JPEG is a compressed image format and therefore it is possible that the size in byte of the compressed file is smaller than the amount of pixels of the image. Sending one byte per pixel will then take longer than sending the compressed JPEG file. For that reason method three should only be considered if the sum of file sizes in byte of the images is greater than the sum of pixels of them.

Obviously, the data to send to the server increases when increasing the number of images to stitch, as shown in Figure 10. The data returned from the server to the client is affected as well by the number of images to stitch.

When using method one the final stitched image will be sent back. So $d_{toClient}$ of method one equals the size of the image result of the stitching process. This size of course depends on the size of the images used to generate it, but also on the amount of overlapping parts of the different images. Less overlapping parts lead to a larger combined image.

The returned XML data of method two or three is much smaller than $d_{toClient}$ of method one, as can be seen in Figure 11. This is because the XML data only contains the homography information for mapping images. The amount of homography information needed for one image in the warping process is approximately constant, therefore the size of the XML data increases linearly when adding more images.

The transmission time needed to send all images to the server and the resulting image back to the client, as done by method one, was the highest of all the methods. Although method two also sends all image data to the server, only the XML data is returned to the client, which results in a shorter transmission time, as shown in Figure 12. Method three leads to the smallest transmission time, if the sum of pixels of the images is less than the sum of file sizes in byte.

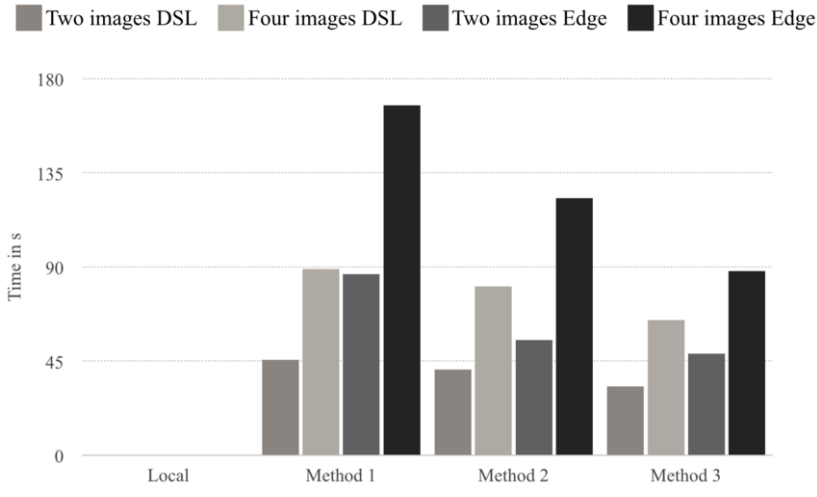

Fig. 12. Transmission Time t_{trans}.

6.5 Memory Usage

The memory usage of all offloading methods is less than the memory usage of the completely local execution, as shown in Figure 13. Obviously, memory usage is independent of network conditions, but it increases when adding more images to stitch.

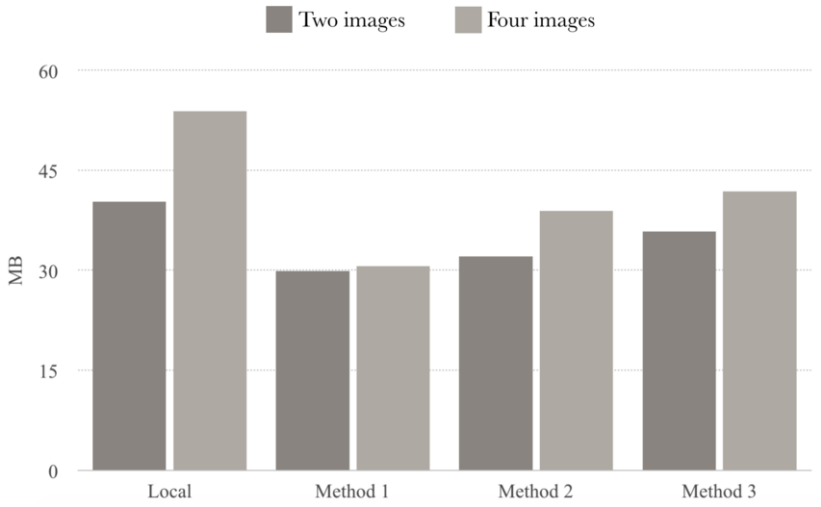

Fig. 13. Memory Usage m.

As all the stitching computation has been migrated to the server, method one needs the lowest amount of memory in all cases. The memory usage decreased by up to 25% for two images and up to 45% for four images when using method one. Method three has the highest memory usage in comparison to the other offloading methods, because both the steps of image compression and warping and projection are executed on the mobile device.

6.6 Discussion

Under the conditions in our experiments all offloading methods result in decreased total execution time in comparison with the completely local execution, if the network condition is at least Edge. As $T_{local} > T_{offload}$ is always satisfied, it can be said in general that using the offloading methods is beneficial. Depending on the network speed and latency, the total execution time could be decreased by around 50% for stitching two images and 73% for stitching four images together.

Contrary to the presumption of Section 4.4, method one results in the smallest total execution time and local execution time of the offloading methods in nearly every case, which makes this method clearly be the best. To trade transmission time for local execution period only makes sense for stitching two images under bad network conditions.

Using method one it was even possible to stitch high-resolution images. The images are directly sent to the server as files and the result is directly written into

a file. Therefore the images do not have to be loaded into the application and no out of memory exceptions happened.

However, the transmission time is the highest when using method one. If the size of data to transmit should be minimized, method two is preferrable. In some cases as the sum of the file sizes of the images in byte is greater than the sum of pixels of the images, method three is optimal, because it needs the smallest data transmission time.

Any offloading method decreased the memory usage in comparison to the memory usage of the local execution. The memory usage could be decreased by up to 45%. It is minimized by using method one. In general method one will be the best choice to minimize the total execution time and relieve the mobile device and therefore improve the user experience.

7 Conclusions

In this paper we have shown that mobile application offloading can improve the performance of image stitching on mobile devices. We have investigated and compared three different strategies to perform offloading.

The measurements in Section 6 show that image stitching on mobile devices benefits of the use of mobile offloading. All of the proposed offloading methods are leading to decreased total execution time, local execution time and memory usage compared to the completely local execution on the mobile device and therefore improve the user experience.

Perhaps the measurements should be repeated with a more optimized version of the application and with more diverse image sets. We conclude that all described offloading methods for image stitching aim at finding a good tradeoff between local execution time and transmission time. It has turned out that sending all the computation to the resourceful and powerful remote server is the best choice in terms of the total execution time and that it even enables us to stitch high-resolution images without any optimisation of the program regarding the execution on mobile devices.

However, the developed program for stitching images is not optimized yet. Parallelization is not used, but possible at many stages of the stitching process. The features of the different, independent images could be found in parallel, to give an example. It is also possible to reduce the complexity of the image stitching algorithm by adding some preconditions, like the order of the images, as described by Herbon et al. [13]. The results received by the stitching process could be further improved by techniques like filtering, feathering or blending [16], to hide the transition from one image to another.

The offloading methods described in this paper are three good working, but basic ones. They are only a sneak preview of the possibilities mobile offloading can offer. The offloading methods can be extended in many ways and adding decision making, like choosing the offloading method dynamically according to network speed, number of images and image size, and local restart methods described in [26] could lead to further improvements.

References

[1] AutoStitch. http://appcrawlr.com/ios/autostitch-panorama/.

[2] DMD Panorama. http://appcrawlr.com/android/dermandar-dmd-panorama/.

[3] Jama-library. http://math.nist.gov/javanumerics/jama/.

[4] Anubhav Agarwal, C. V. Jawahar, and P. J. Narayanan. A survey of planar homography estimation techniques. *Centre for Visual Information Technology, Tech. Rep. IIIT/TR/2005/12*, 2005.

[5] Nitin Bhatia and Megha Chhabra. Accurate corner detection methods using two step approach. *Global Journal of Computer Science and Technology*, 11(6):24–30, 2011.

[6] Matthew Brown and David G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

[7] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[8] Robert Collins. Lecture 06: Harris corner detector, 2007.

[9] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[10] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[11] C. Harris and M. Stephens. A combined corner and edge detector. In C. J. Taylor, editor, *Alvey Vision Conference 1988*, pages 23.1–23.6, 1988.

[12] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, UK and New York, 2nd ed. edition, 2003.

[13] Christopher Herbon, Klaus Tönnies, and Bernd Stock. Adaptive planar and rotational image stitching for mobile devices. In Roger Zimmermann, editor, *the 5th ACM Multimedia Systems Conference*, pages 213–223, 2014.

[14] Dijiang Huang, Xinwen Zhang, Myong Kang, and Jim Luo. Mobicloud: building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 27–34. Ieee, 2010.

[15] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.

[16] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *Computer Vision-ECCV 2004*, pages 377–389. Springer, 2004.

[17] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[18] Pablo Márquez-Neila, Javier López-Alberca, José M. Buenaposada, and Luis Baumela. Speeding-up homography estimation in mobile devices. *Journal of Real-Time Image Processing*, 2013.

[19] H. P. Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Stanford University. Computer Science Department, Stanford, 1980.

[20] Ofir Pele. Sift - the scale invariant feature transform, 2009.

[21] Masatoshi Sakamoto, Yasuyuki Sugaya, and Kenichi Kanatani. Homography optimization for consistent circular panorama generation. In *Advances in Image and Video Technology*, pages 1195–1205. Springer, 2006.

[22] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[23] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[24] Richard Szeliski. *Computer vision: Algorithms and applications*. Texts in computer science. Springer, London and New York, 2011.

[25] Qiushi Wang, Marti Griera Jorba, Joan Martinez Ripoll, and Katinka Wolter. Analysis of local re-execution in mobile offloading system. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 31–40. IEEE, 2013.

[26] Qiushi Wang and Katinka Wolter. Reducing task completion time in mobile offloading systems through online adaptive local restart. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, pages 3–13, New York, NY, USA, 2015. ACM.

[27] Huaming Wu, Qiushi Wang, and Katinka Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 ICC - 2013 IEEE International Conference on Communication Workshop (ICC)*, pages 728–732, 2013.